

CLEAR

Ray Ozzie

January 2017

Why am I interested in this issue?

- The crypto wars of the 90's left me with a high degree of sensitivity related to crypto issues.
- I'm concerned by the extremely polarized characterizations of an inherently nuanced issue, and dangerously entrenched postures.
- I do not believe that the issue will go away.
- If I can, I'll do what I can to help avert a highly-charged showdown.

Why does industry take such a strong posture?

- Execs are generally mature, well-intentioned, and open-minded.
- But their products are under constant attack
“nobody understands how difficult it is to build secure systems that scale”
- They must develop homogeneous products for the global market
“nobody understands how difficult it is to deliver services to billions of people globally”
- Many, some immigrants, worry about unintended consequences globally
“I can’t feel complicit in enabling mass-surveillance, or in violating human rights”
- The risks feel unbounded; post-Snowden many felt publicly defensive
“I won’t even entertain any notion that might put security, business, reputation at risk”

But perhaps a bit of confirmation bias?

*“Security experts agree that it’s **not possible** to give government what it wants without creating vulnerabilities exploitable by bad actors. If the FBI can eavesdrop on your conversations, so can cybercriminals. So can the Chinese. So can terrorists”*

*“It’s not possible. Math can’t be negotiated away just because it’s inconvenient. **Denialism** is a deadly feature of 21st-century life.”*

*“The government cannot wish away representations by cryptographers and industry leaders regarding the limits of technology using **magical thinking**.”*

* quotes edited for brevity

The Foundational Observations

1. The largest vendors already have a deeply-rooted core competency in secure *key management*, because they need to do *software update*.
2. There is natural and sustainable alignment around this competency; this is not something they do half-heartedly or begrudgingly:
 - Product quality and customer satisfaction depends upon it
 - BoD and management are accountable for the business continuity tied to it
 - Ultimately, market cap and shareholder interest are tied to it
3. The largest vendors have large investments in trained staff (lawyers, engineers) to vet & comply with gov't processes on a worldwide basis.
4. Large vendors are resourced at a level that they can act, for citizens, as an effective balance to the power and resources of world governments

CLEAR

In early 2016 I began working on this to see if we could viably raise the level of discussion to “should we” (not “can we”) technically implement secure EA.

Given my background, I have a very strong belief that the greatest impact (business, society) arises from behavior & *default settings* of only the most popular (i.e. 250M+ DAU) devices, software, and services.

- For devices, there are only a handful of vendors that have scale and matter
- For software, it is impractical and unadvisable to mandate anything that impacts research or startup innovation. Even China can't control every app. (And not even necessary: new products tend to be most readily exploitable.)

CLEAR

The method I've developed for devices isn't the only possible solution, but it

- Is readily implementable with little more than an OS software upgrade
- Doesn't require secret algorithms or backdoors, new math to be proven, "chip off" capabilities, or hardware redesigns of any kind
- Doesn't require any government keys, new keyholders, or 3rd party keyholders
- Doesn't require any new processes for interactions between vendors & govt beyond what's done today for cloud data, and done internationally via MLATs
- Although nothing is risk-free, it doesn't introduce materially new *classes* of risks *beyond those that large-scale vendors already need to cope with.*

CLEAR uses a *public key*, analogous to the one already distributed by each vendor on its devices for *software update*, to securely “wrap” a secret that (only) the device vendor might later use to unlock a device.

Each device can have a *secret key* that unlocks access to storage/device



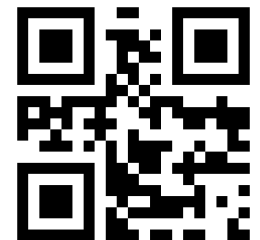
Each device can have a vendor’s *public key*, usable as a secure “*encryption envelope*”



This *encryption envelope* can only be opened by the vendor

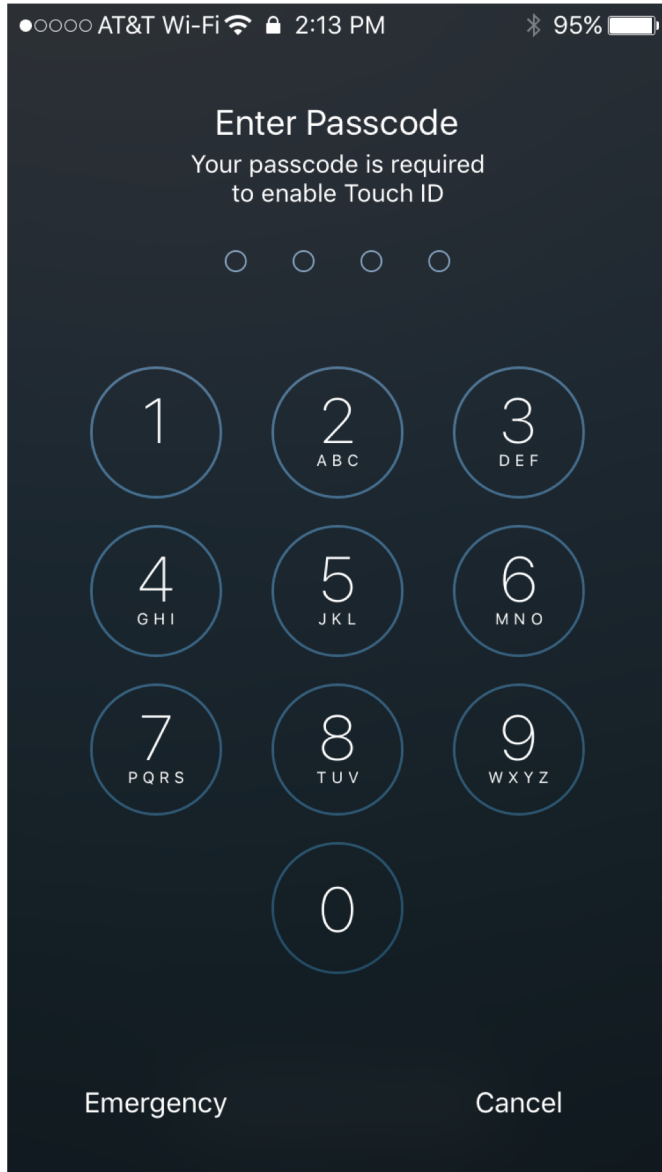


This *envelope* can be formatted for easy transport to the vendor



(technical notes)

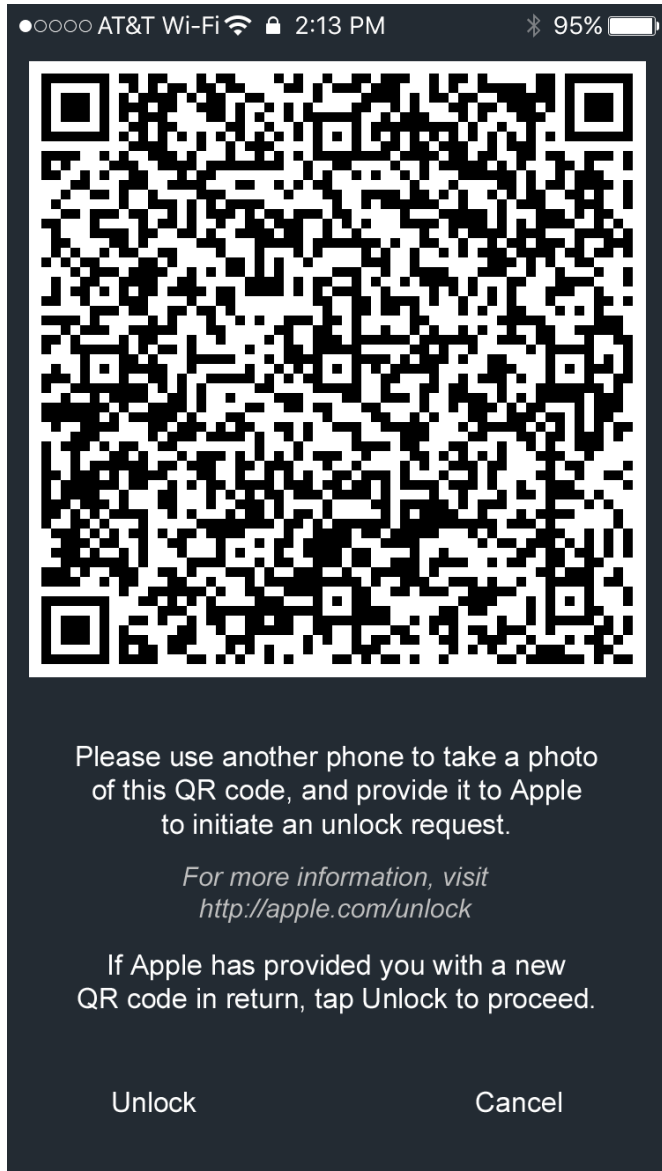
- When the device is first unlocked after the initial OS upgrade, it generates what is in essence a “vendor PIN” (or “vendor fingerprint”) that could be used to unlock the device. *This is highly implementation-specific, but for the sake of discussion you might think of this as a random symmetric key that wraps the storage protection key.*
- This “vendor PIN” is generated in-memory, encrypted with a public key provided by the vendor, and then the cleartext version is securely discarded.
- This “vendor-encrypted vendor PIN” (referred to later as a “device unlock request”) would then be retained in NVM in an area available when the lock screen is active.
- This “vendor-encrypted vendor PIN”, being encrypted, is not a secret.
- If this “vendor-encrypted vendor PIN” is deleted or corrupted via jailbreak, the worst case is that this device is no longer available for EA by government.



In this example, government is in possession of a locked phone, tablet, PC, or other device, and has obtained a legally-authorized warrant permitting access to the contents of said device.

The vendor in this example would have implemented a new gesture at the lock screen – perhaps a multi-finger swipe or twist gesture - to gain access to a special “Device Unlock” screen shown on the next slide.

(Vendors should be motivated to discuss this behavior quite openly and transparently with their customers; privacy issues are deeply important and this shouldn't be a secret.)

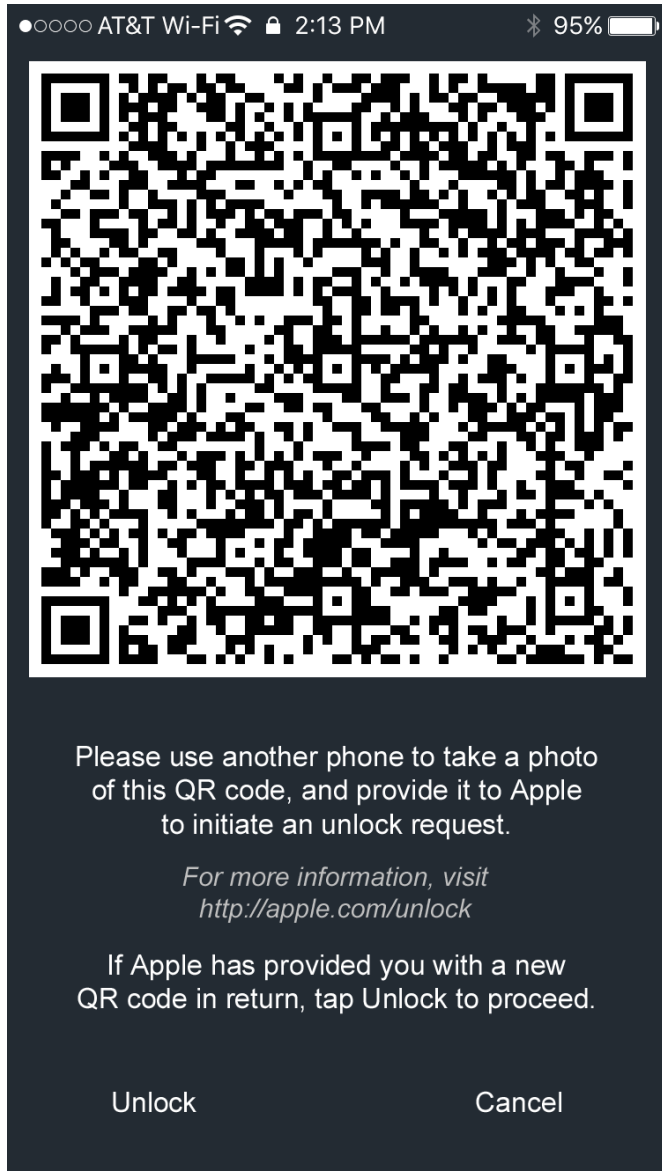


Upon performing the gesture, this new screen is shown. What you see here is the “device unlock request” envelope, securely encoded and displayed as a QR code.

This “device unlock request” data, previously encrypted by the vendor’s public key includes

- The symmetric encryption key that would be usable so as to unlock the phone and its storage
- Information that uniquely identifies the hardware, including its IMEI, WiFi MAC, and BT MAC addresses

Due to the benefits of strong cryptography, the data encoded within this “device unlock request” cannot be decrypted by anyone other than the vendor.



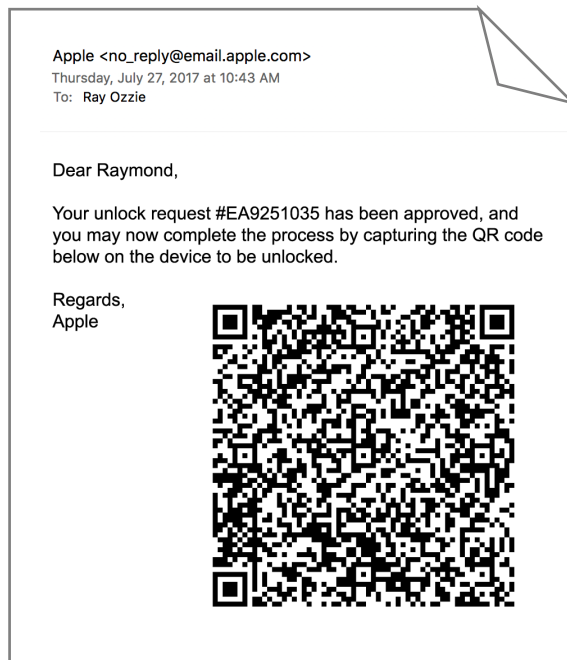
Government would then use a separate phone to take a photo of this screen, and would then include that photo as a part of the warrant documents issued to the vendor.

While this screen is being displayed, the OS software would also ensure that the phone is continuously broadcasting a simple “ping” message on both WiFi and BT, using the hardware’s real MAC addr’s.

Government would then use any commonly-available software scanner apps to observe the WiFi and BT MAC addresses, and would also include these in the warrant documents issued to the vendor.

By providing MAC addresses that are “observed”, and which are also encrypted in the QR data, the vendor can verify this “Proof of Possession” of the device.

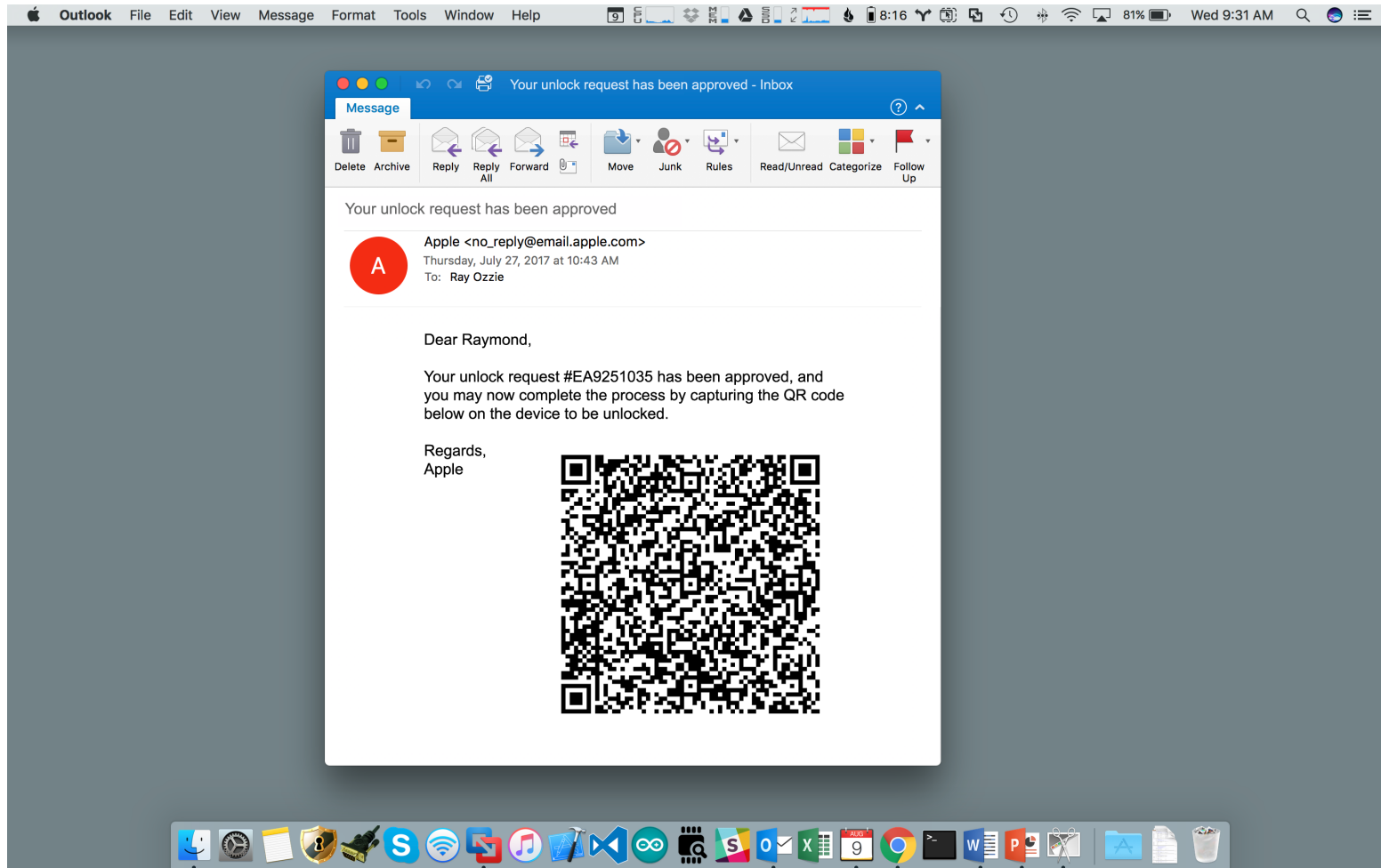
(Alternatively, a separate iPhone app could be provided by the vendor to governments that would extract this barcode data via BT, and which would also do an active cryptographic challenge to enhance assurance of Proof of Possession.)



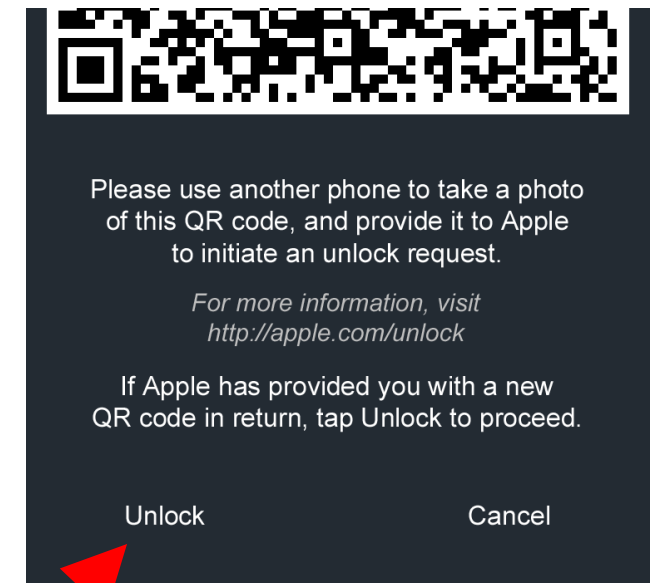
Upon receiving the warrant containing the QR code photo, the vendor's attorneys would first scrutinize the request.

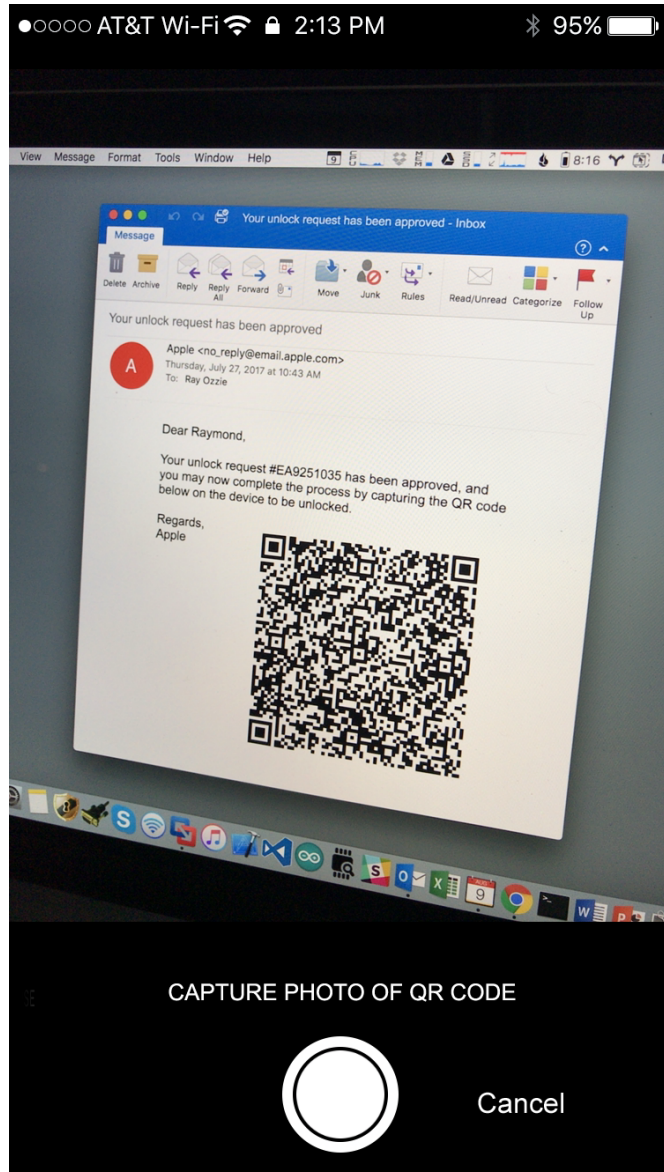
Iff deemed actionable, the vendor then would use its existing secure HSM facility and apparatus to decrypt the key within the QR code, exposing the "unlock key" for that (and only that) device.

This, along with warrant/audit information and *unlock policies*, are then re-encrypted by the HSM and encoded as a new QR code, to be returned to government, i.e. by email.



Upon receiving email from the vendor, the government taps the Unlock button on the device.





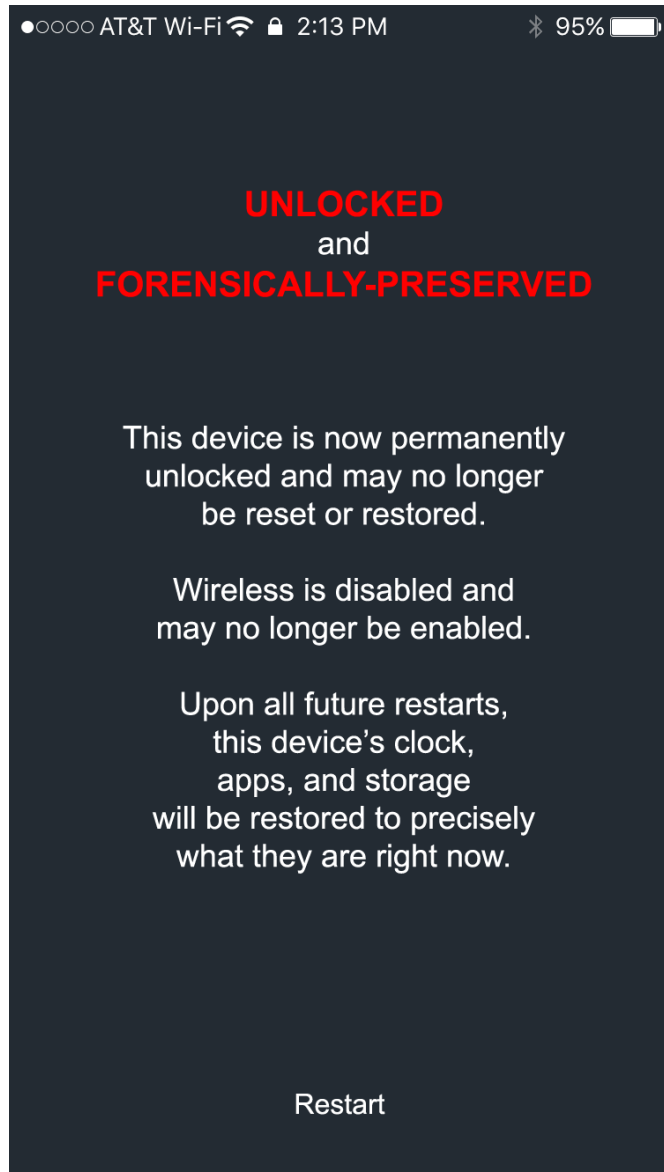
This brings up the device's built in camera, in a special mode that enables them to capture of the QR code.

If the QR code is not valid for any reason, the device will not accept it, and the device will fail to unlock.

Note that no QR code will be recognized as valid unless it came directly from the vendor, and any QR code provided by the vendor will cryptographically only work on the single device that originated the request.

If it was indeed a valid QR code provided by the vendor, the unlock response is decrypted using the vendor's public key, and the device is then unlocked according to the contained *unlock policies*.

The next screen will then be displayed.



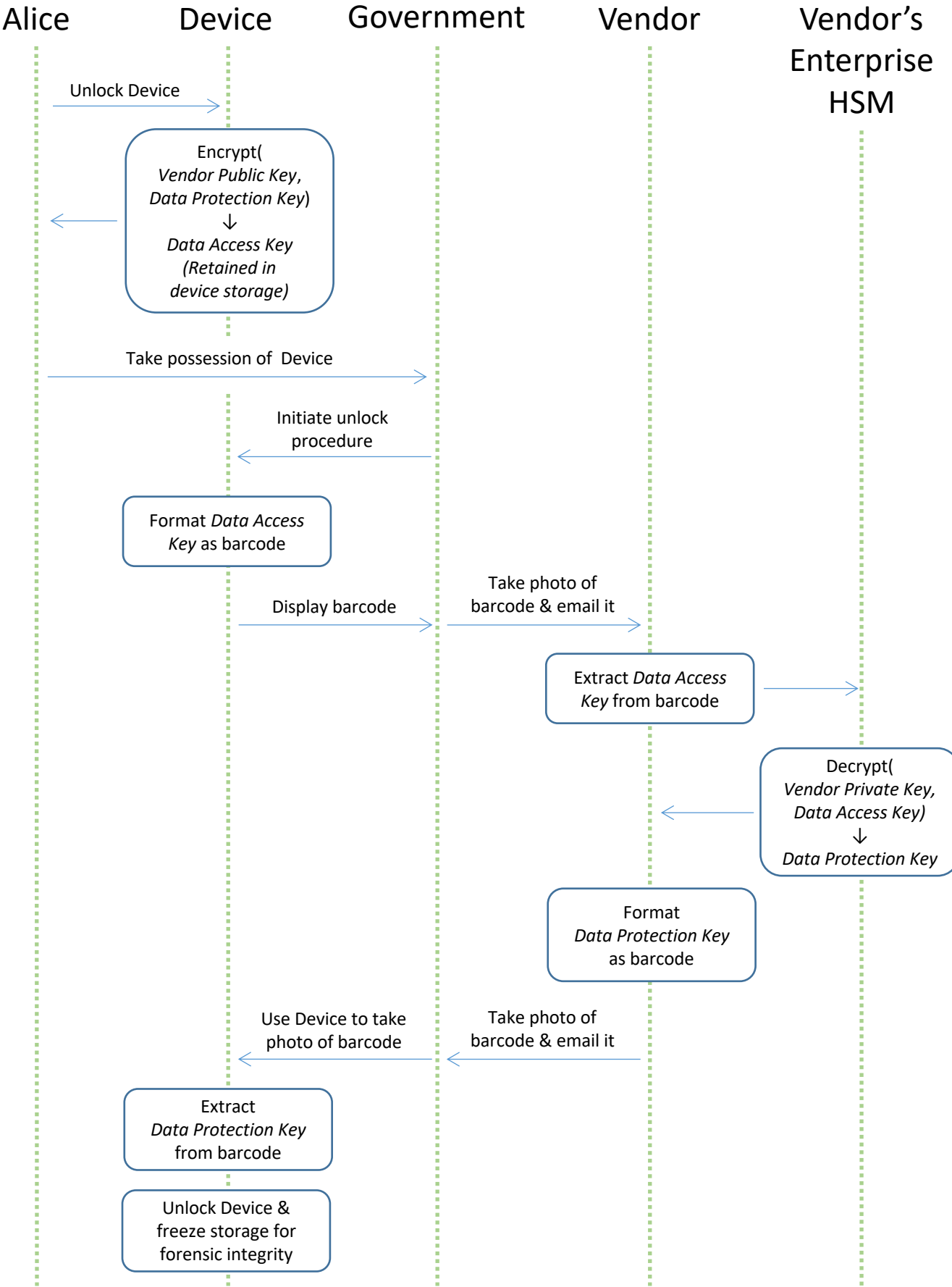
And that's that. Since this was an unlock request made in response to a warrant, it is most useful for the OS to then place the phone into a forensically-secure state because this is legally favorable to *both* government and the subject.

The *unlock policy* that placed the phone into the forensically-secure state would cause internal JTAG fuses to be blown irreversibly, or using SOC-specific techniques such as the RDP Level 2 protection on the STM32.

Depending upon the implementation, additional warrant/audit information encoded into the vendor response would be display on this screen.

It's important to note also that this is a “dual-use” methodology and there is nothing preventing a vendor from using this same method with different *unlock policies* for other types of simple unlock requests, whether from business or the family of the deceased.

- I claim that this “vendor-authorized EA” procedure demonstrates that *there is at least one technical method that does not materially increase cybersecurity risk*, even against state-level adversaries.
- I claim this can be implemented by a vendor in a manner with manageable risk.
I do not claim that this method is without risk, the most likely threat vector being volume-related “people risks” that (for a single device) might somehow bypass audits within the vendor’s authorization processes.
 - Today, by necessity, vendors authorize thousands of their internal developers to sign code on a daily basis by using alternate certificates chained to the HSM-managed root keys. It is reasonable to assume that vendors would similarly design and implement strongly-audited custom practices such as key rotation, per-manufacturing-run intermediate certs, and so on. *This is highly vendor-specific* for both technical and internal process reasons.
 - Software Update is a very high-risk technology and practice. If it were being proposed today, no sane person would believe that it could be implemented at scale. In theory, it just can’t work. But it works in practice because of the strong economic and business need for it to exist. Software update *has* exhibited critical vulnerabilities, and there are no vulnerabilities that are more rapidly addressed.
- I do claim that of any possible EA methods, *vendor-authorization* is likely *optimal* because of pure alignment: Even with no internal need for EA in their business, the vendor is nonetheless self-motivated to design trustworthy, economical, scalable processes & technologies that are appropriate for *their own* product, engineering and operations contexts. Any solution imposed or involving third parties would introduce risks that they themselves could not manage.



Defense in Depth: Crypto Vulnerabilities

- Although nothing is risk-free, high-quality HSMs *will* for all practical purposes prevent any disclosure of private keys
- Still, there may be vulnerabilities over time
 - Cryptanalytic advances may impact choices of key lengths or algorithms
 - Weaknesses may be found in random number generator implementations
- Implementations should provide high assurance that encrypted data recorded in the past shouldn't then suddenly become decryptable
- User privacy must be paramount. As such, on an HSM failure, the worst case should be that no more EA can be performed until new keys and algorithms have been widely deployed

Risk Reduction thru Redundancy **

- One way to approach this issue is through use of “two of everything”
 - Two symmetric algorithms/parameters (ie AES & Blowfish), plus
 - A unique TRNG/whitening technology (ie quantum & noise) paired w/each
 - Two asymmetric algorithms/parameters (ie ECC & RSA), plus
 - A unique HSM (ie Gemalto & Thales) paired with each
- It is very important to ensure the simplicity of the algorithms - retaining crypto semantics while performing sequential super-encryption within
- The goal is that newfound vulnerabilities discovered in any one will mean that existing encrypted data is still fully protected by the other

** Note that this approach is not specific to CLEAR or EA. It originated in designs to protect Software Update keys.

Symmetric Key Operations

App.NewRandomKey() would be implemented as:

```
Marshal(Alg1.Keygen(RNG1), Alg2.Keygen(RNG2))  
→ Key
```

App.Encrypt(Data, Key) would be implemented as:

```
Unmarshal(Key) → Key1, Key2  
Alg2.Encrypt(Alg1.Encrypt(Data, Key1), Key2)  
→ Encrypted Data
```

App.Decrypt(Data, Key) would be implemented as:

```
Unmarshal(Key) → Key1, Key2  
Alg1.Decrypt(Alg2.Decrypt(Data, Key2), Key1)  
→ Decrypted Data
```

Asymmetric Key Operations

Vendor.NewPublicKey() would be implemented as:

```
Marshal(HSM1.Keygen(Alg1).PubKey1, HSM2.Keygen(Alg2).PubKey2)  
→ VendorPubKey
```

App.Encrypt(Data, VendorPubKey) would be implemented as:

```
Unmarshal(PubKey) → PubKey1, PubKey2  
Alg2.Encrypt(Alg1.Encrypt(Data, PubKey1), PubKey2)  
→ Encrypted Data
```

Vendor.HSM.Decrypt(EncryptedData) would be implemented as:

```
HSM2.Decrypt(HSM1.Decrypt(EncryptedData))  
→ Data
```

let's talk.